Updates on the OSDF Monitoring System

Derek Weitzel

Motivation

- The Open Science Data Federation supports the data needs of organizations and individual users
- Caches and origins are spread throughout the world (see right →)
- Monitoring cache usage is imperative:
  - Working set size
  - Cache thrashing
  - Utilization

https://osdf.osg-htc.org

Summary

- We completed 2 validations, a correctness and scale
- We found a few minor issues, which we corrected
- Found a large problem with the transport mechanism between the XRootD servers and our accounting collector
    - Solved by writing a simple shoveler that reliably transports monitoring/accounting packets between the XRootD server and our accounting pipeline

OSDF Monitoring

- The OSDF is built off of XRootD file server software

OSDF monitoring = XRootD monitoring

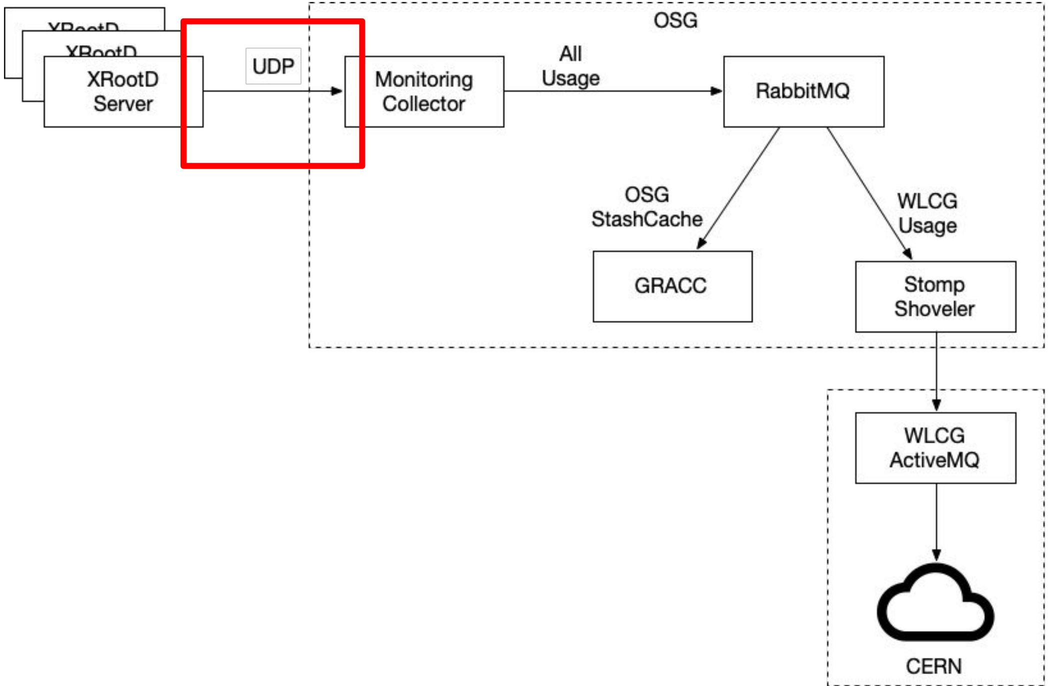
Validations

We conducted 2 validations of the existing OSDF/XRootD monitoring to find and correct any issues.

- Correctness Validation: Aug 12, 2020
  - Is every transfer captured correctly
- Scale Validation: Apr 14, 2021
  - Does our monitoring scale to the expected size of the OSDF in the future.

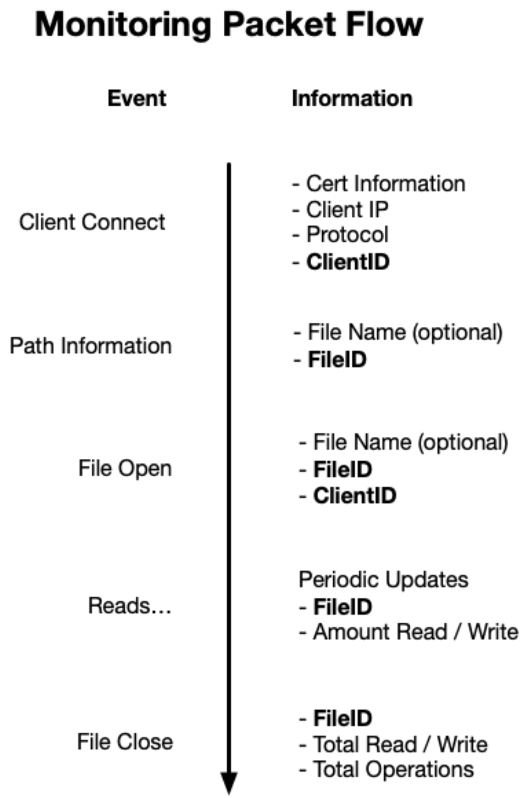
XRootD Monitoring = XRootD Detailed Monitoring

- Current monitoring uses detailed collector packets



Why XRootD Detailed Monitoring is Hard - Format

- Collector has to keep a lot of state
- Potential for packet loss means we have to place TTL on state
- Time between client connect and file close can be **hours**
- Must “join” different messages, but may lose packets
- For example, if you get a file close without the corresponding file open, then no idea what file was read.



Observations from validation v1

- Small bugs in Collector
- Incorrect assumption: Sequence numbers in monitoring packets are not a reliable measure of missed packets (since fixed)
- **UDP fragmentation caused significant loss**

Report: https://doi.org/10.5281/zenodo.3981359

UDP Fragmentation

- UDP Fragmentation is a known problem: https://blog.cloudflare.com/ip-fragmentation-is-broken/
- The very Zoom meeting you are on uses UDP packets:

```
  0100 .... = Version: 4
  .... 0101 = Header Length: 20 bytes (5)
▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
  Total Length: 1092
  Identification: 0xddbb (56763)
▼ Flags: 0x4000, Don't fragment
    0... .... .... .... = Reserved bit: Not set
    .1.. .... .... .... = Don't fragment: Set
    ..0. .... .... .... = More fragments: Not set
  Fragment offset: 0
  Time to live: 41
  Protocol: UDP (17)
  Header checksum: 0x558f [validation disabled]
  [Header checksum status: Unverified]
  Source: 198.251.146.181
  Destination: 192.168.0.5
```

Tests performed in validation 2

In the second version of our validation we wanted to find out:

1. If sending monitoring data simultaneously from multiple XRootD servers would show any kind of data loss.
2. What is the maximum rate at which our collector can process monitoring records.

Monitoring data from multiple XRootD servers

On each test a client will request ‘N’ number of random files to each of the ‘M’ servers, then wait for a second and repeat until a total amount of ‘O’ files is reached where:

N - Req. rate
M - Num. Servers
O - Total files req.

After each test. we will pull the recorded data from rabbitMQ and compare with what we requested.

With this experiment we concluded that data loss due to scale is negligible

Num. Servers	Files req. per server	Total files req.	Req. rate	Files recorded avg.	Success %
2	100	200	20/s	200.00	100.00%
4	100	400	20/s	400.00	100.00%
8	100	800	20/s	800.00	100.00%
32	100	3,200	20/s	3196.67	99.90%
50	100	5,000	20/s	5000.00	100.00%
50	200	10,000	50/s	10000.00	100.00%
50	400	20,000	80/s	19992.33	99.96%
50	800	40,000	100/s	39991.00	99.98%

Summary of major issues

- Fragmentation causes **loss** of packets leading to missing data
- When scaling the number of nodes and the number of packets, packet loss occurs.

Solution - XRootD Monitoring Shoveler

- Designed and develop a “shoveler” from the UDP format to a resilient format (Message Bus)

- The shoveler is simple, does no parsing or aggregation of records:

Shoveler Operation

1. Receives Packets
2. Very simple validation
3. Packages the data packet (base64’s the data, puts in json with other metadata)
4. Reliably sends to message bus

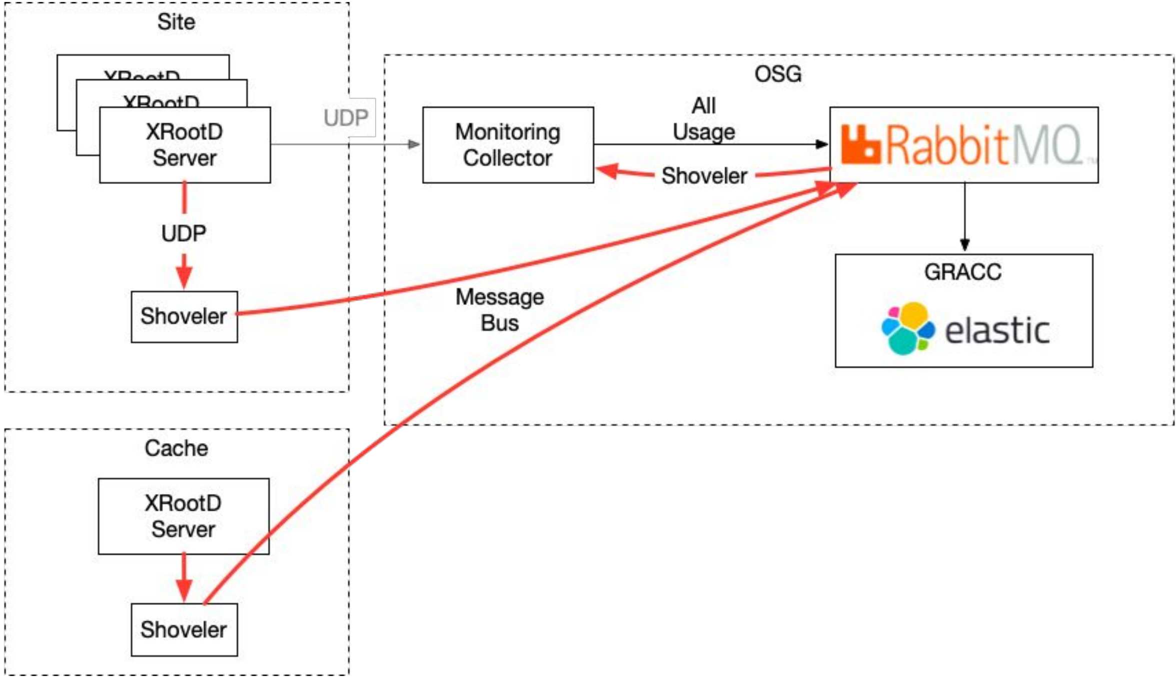
XRootD Monitoring - 2 components

- **Shoveler (simple):**
  - **Runs at Sites**
  - Collects the monitoring UDP packets from XRootD
  - “Packages” the UDP messages and sends them to a reliable message bus

- **Collector (complicated):**
  - **Runs Centrally**
  - Parses monitoring messages
  - Keeps state
  - Processes packets to extract VO, application info, type of transfer

Shoveler

- A lightweight shoveler from UDP to a resilient transfer method
- Connection to RabbitMQ
- Caches will run shoveler "side-car"



Design Decisions

- The shoveler is purposefully “simple”
- The collector performs all stateful logic

- When shoveler is disconnected from message bus, it will write messages to disk and replay them when reconnected.
  - A production shoveler will write ~30MB of data a day to disk if disconnected.

Shoveler

Available at
https://github.com/opensciencegrid/xrootd-monitoring-shoveler/releases

Will be available in OSG’s repos soon (currently in OSG testing)

Can be deployed as a static binary, RPM, docker image, or in kubernetes.

Deployment plan

- The shoveler has been deployed at several sites.
- Next, we will deploy the shoveler as a “side-car” with the distributed caches of OSDF
- Shoveler will become part of the XrootD deployment

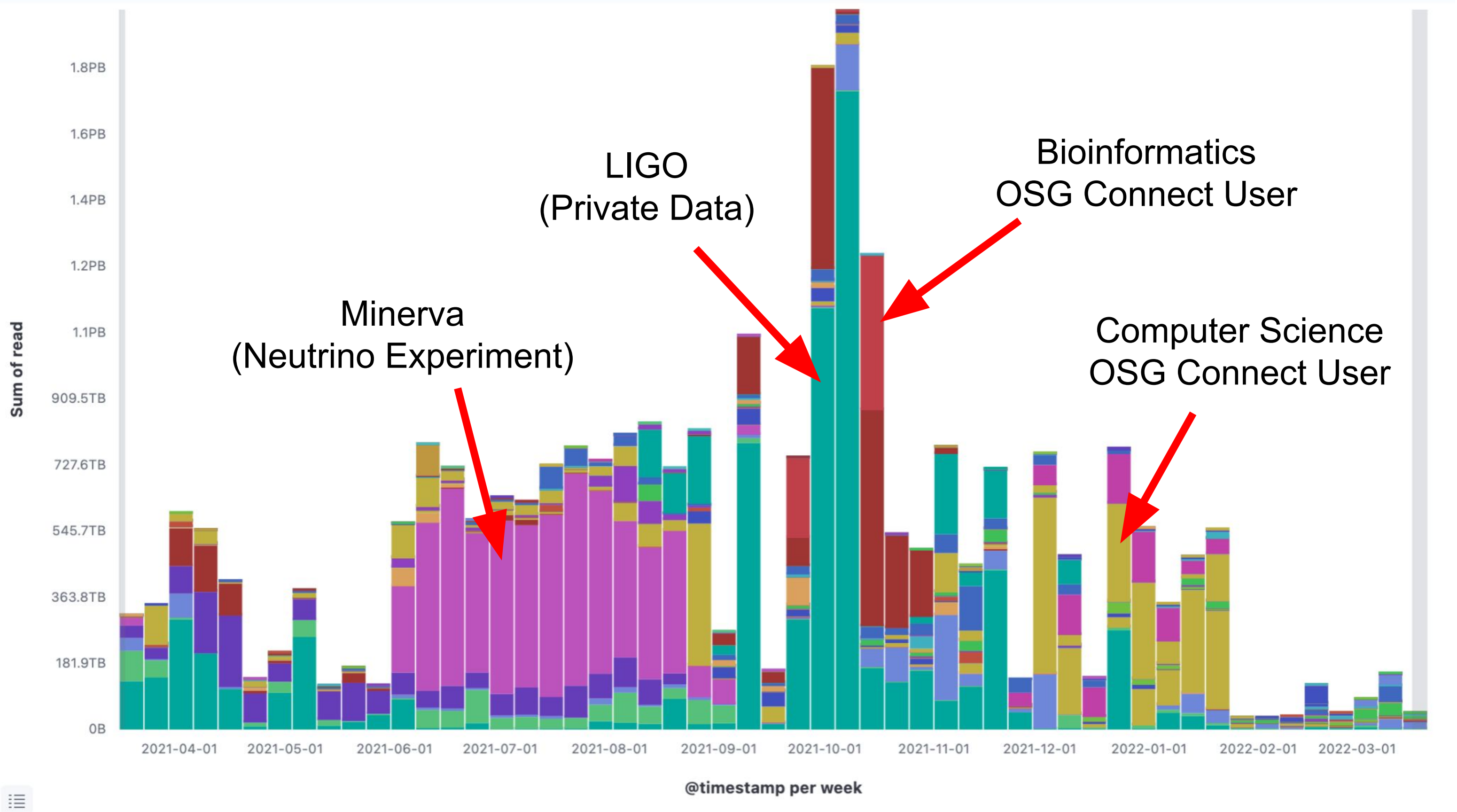
Statistics of the OSDF

Busiest Cache

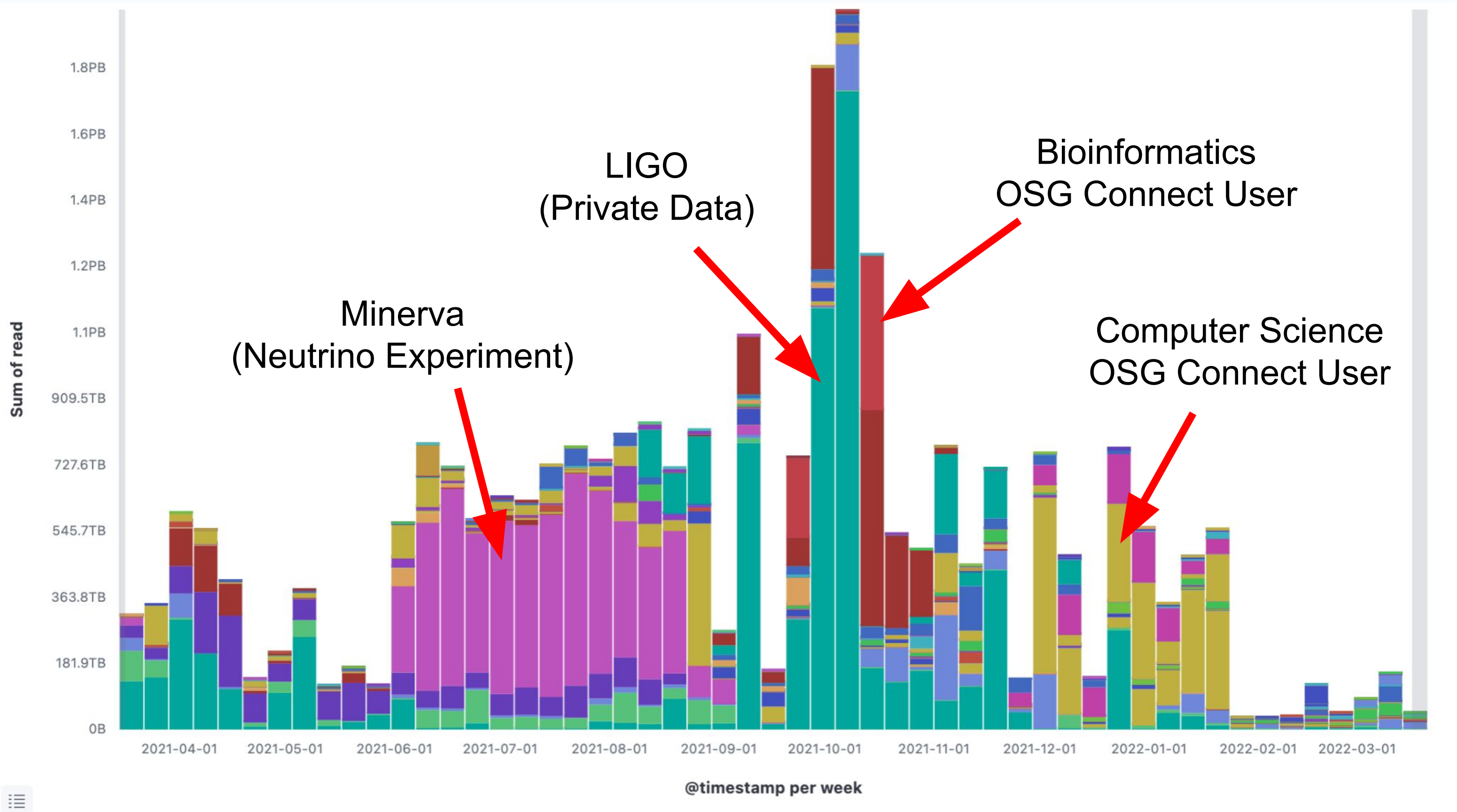
CHTC Cache → tiger0002.chtc.wisc.edu

Cache / Origin	Data Read
osg.chic.nrp.internet2.edu	6.2PB
osg.kans.nrp.internet2.edu	4.6PB
tiger0002.chtc.wisc.edu	4.3PB
osg.newy32aoa.nrp.internet2.edu	2.9PB
its-condor-xrootd1.syr.edu	1.6PB
stashcache.t2.ucsd.edu	1.3PB
osg.sunn.nrp.internet2.edu	874.5TB
fiona-r-uva.vlan7.uvalight.net	760.1TB
osg.hous.nrp.internet2.edu	621.5TB

OSDF usage by Week for last 1 year



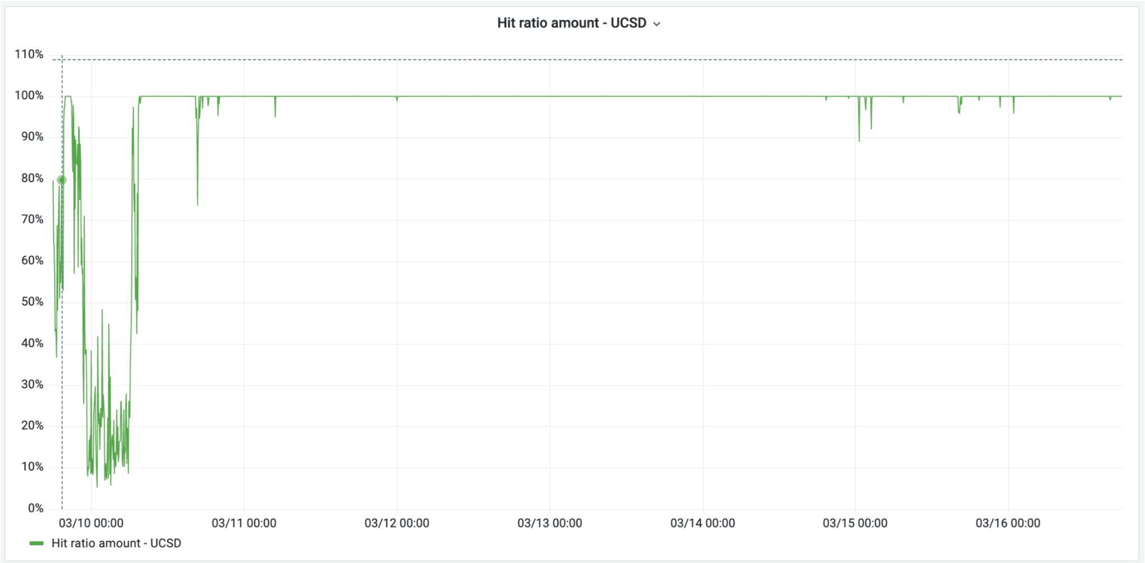
92 researchers, 9 collaborations & 1 campus!



New Developments

XRootD added a “g-stream” which includes caching information such as hit rate and bytes read from origin vs. cache.

We will be integrating this more with our data pipeline and visualization



Summary

- We completed 2 validations, a correctness and scale
- We found a few minor issues, which we corrected
- Found a large problem with the transport mechanism between the XRootD servers and our accounting collector
    - Solved by writing a simple shoveler that reliably transports monitoring/accounting packets between the XRootD server and our accounting pipeline

Acknowledgments

This project is supported by the National Science Foundation under Cooperative Agreements OAC-2030508 (PATh) and OAC-1836650 (IRIS-HEP). Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.